PRODUCT DOCUMENT · TESTING AND CODE REVIEW

Testing and review: how a change earns its merge

Proof from the implementer, an independent reviewer from another vendor, a tester for the whole piece of work, and a person who approves every merge. How it works, and what it produced on one large project.

PUBLIC EDITION

789	23%	1.6	0
review verdicts in the record, 5 Nov 2026	of reviewed changes sent back	lines of test per line of Go production code	merges an agent can make

MEASURED ON

one large agent-built project

main branch · 5 November 2026

PUBLISHED

5 November 2026

Saphan Studio · saphan.ai/product

CONTENTS

01 Summary

02 The path of one change

03 Proof, not declaration

04 Tests written apart from the implementation

05 The review: five acts

06 Who reviews whom

07 The tester: the whole train, from outside

08 The merge gate

09 Measured from the record

10 The same chain, as open-source tools

11 Code quality against published baselines

12 What this document does not claim

13 The Saphan ecosystem and this document set

HOW TO READ THIS DOCUMENT

Section 1 is the summary. Sections 2 to 8 describe the process: the path of a change, the proof, tests written apart from the code, the five review acts, reviewer independence, the tester and the merge gate. Section 9 gives the measured numbers and how each was counted.

SECTION 01

Summary

In Saphan Studio a change reaches the main branch only after three separate checks and one human act:

01 **The implementer proves its own work** with evidence, not a statement: a failing test before the fix, the same test green after it, and the numbers of tests run and skipped.

02 **An independent reviewer judges the change** against the order's contract, in five fixed acts. The reviewer never wrote the change and, by default, comes from a different AI vendor.

03 **At the end of a larger piece of work, a tester checks the whole** from the outside, as its user would, on its own disposable instance.

04 **A person approves the merge.** No agent can merge. The engine refuses it.

Section 9 shows what this produced on a large software project built by AI agents under Saphan Studio, measured on 5 November 2026: of 789 review verdicts, **183 (23%) sent the work back** and 346 (44%) accepted it only with conditions.

MEASURE	VALUE	HOW IT WAS COUNTED
Review verdicts in the record	789	accept, accept with conditions, reject (section 9)
Sent back by the reviewer	183 (23%)	verdict "reject"
Lines of test per line of production code	1.6	Go, main branch, generated code included
Merges in the last seven days	280	main branch

SECTION 02

The path of one change

`signed order` → `plan (optional)` → `implement in own copy` → `proof` → `independent review` → `human approval and merge` → `closure`

STEP	WHO	WHAT MUST EXIST BEFORE THE NEXT STEP
Order	planner, signed by a person	The scope, what is not asked for, acceptance as numbers with the command that measures them, the skills to use
Plan	implementer	For larger work: a plan the planner and a reviewer can check before any code (can be skipped for small work)
Implement	implementer, in its own working copy	Commits in its own branch; it cannot touch the main branch or another agent's copy
Proof	implementer	A status file with the last commit, every acceptance item and its evidence; missing file = incomplete return
Review	an agent that did not write the change	A verdict: accept, accept with conditions, or reject, with the location and remedy of every finding

STEP	WHO	WHAT MUST EXIST BEFORE THE NEXT STEP
Approval and merge	a person	The person’s act, confirmed with a presence code or passkey; the merge checks the commit rules
Closure	the engine	The working copy is archived and verified, then removed; the record shows the work closed

Work is grouped into **trains** of steps. Each step passes its own review. When the last step has landed, one tester checks the whole train. A train is not closed while a blocking tester finding is open.

SECTION 03

Proof, not declaration

Every order carries the same definition of done. It is not renegotiated inside an order. Its main rules:

- **Proof, not declaration.** “Tests pass” is not evidence. The output of the command, at a named commit, is.
- **Red first.** A defect is reproduced by a test that fails on the base for the reason the order states, before any fix. After the fix the implementer reverts it once and watches the test fail again.
- **Baseline first.** The implementer measures the suites before changing anything, so a failure that was already there is not blamed on the change, and a new one is not hidden among old ones.
- **Tests run and tests skipped are numbers.** A skipped test is reported, never silently dropped. What cannot be run is declared UNRUN with the reason.
- **Green is not correct.** A green suite says the tests passed, not that the change does what was asked. That is why the review exists.
- **A false premise ends the step.** If the order rests on something that turns out untrue, the agent stops and reports. If more must be touched than the order allows, it returns a proposed amendment, not a bigger change.
- **A screen a person uses is not delivered until an agent has opened it**, walked it and tried to break it, on its own instance.

SECTION 04

Tests written apart from the implementation

Tests are not a mirror of the implementation. They are independent work, collided with it.

When the same agent writes the code and its tests, the tests tend to check what was built, not what was asked. So for every change in the security, user-interface or public-interface class, and for every change whose review is named in advance:

01 The planner writes two plans at once: the implementation plan and a **test plan**. Each test case names the change that must turn it red. The two plans share only the contract: names, signatures, endpoints and refusal codes.

02 The contract is cut first, as stubs, in one commit. Both agents start from that commit.

03 A separate **test author** writes the tests. It never sees the implementation or its order. Every test it writes fails at the start, for the reason its case names.

04 The collision happens in the review: the reviewer runs the test author’s tests against the implementation and classifies each red test as a defect of the implementation or a defect of the test. Both land together.

SECTION 05

The review: five acts

Every named review performs the same five acts, in this order. The reviewer recommends; a person decides.

ACT	WHAT THE REVIEWER DOES
A1 · Ground	Fetches the change, finds where it branched from the main line and reads the three-dot diff. If the tip is not the merge candidate, the review stops as unverified.
A2 · Differential	Runs the affected suites and the suites of everything that imports them, at the base and at the tip, on the same machine. Each red is classed new, pre-existing or flaky.
A3 · Unplug	Removes every mechanism the change adds, once. Its test must fail by name. A test that cannot fail is a blocking finding.
A4 · Closure	Adds one case the mechanism was never told about, of the reviewer’s own choosing. It must fail closed, or the fix is the wrong kind.
A5 · Remainder	Checks the change’s own “what this does not catch” paragraph against A3 and A4. An undeclared hole is a condition.

One profile per change

The planner gives each review one profile, chosen by what the change touches, not by its size: **security, backend, user interface, mobile** or **documentation**. Each profile adds its own acts and the instruments of the skills the order names.

Verdicts

- **Accept**: the change meets the contract.
- **Accept with conditions**: the change may land once named, located conditions are met.
- **Reject**: the change goes back with findings. **The second reject ends instance-fixing:** the next round must be a design round, not another patch.

The author’s reasoning never lowers a finding’s class. A re-review judges only the findings and the fix, so a review cannot drift into a new review of everything.

SECTION 06

Who reviews whom

Independence is configured, measured and shown, not assumed. Saphan Studio takes the most independent reviewer available, in this order:

RUNG	REVIEWER	WHEN
1	An agent from **another AI vendor**	the default when another vendor is admitted and free
2	**Another seat** of the same vendor	when only one vendor is available
3	The same seat in a **fresh session**: new context, the review role, never the author's transcript	a single-seat installation
–	**The author's own session never reviews its own work.** There is no fourth rung.	

The rung used is written into the record and shown in the console ("reviewed: same vendor, other seat"), so an auditor sees how independent each review actually was.

SETTING	VALUES	DEFAULT	EFFECT
`review.cross_vendor`	required · prefer · off	`prefer`	*required* refuses by name when no other vendor is free; *off* still produces a real review on another seat or a fresh session
`dispatch.review_seats`	prefer · only · any	`prefer`	*only* keeps reviews on reserved review seats; *prefer* falls back to a free seat and records it

Work that always gets a full-strength reviewer

- **Security** (who may act and what may be reached: grants, egress, signatures, authentication, confinement) gets **two reviews**, one of them on a full-strength model.
- **Network** work and anything **a person reads or clicks** is never reviewed on a low-cost lane.
- Low-cost or free model lanes take small and medium changes, and a change they write is always reviewed by another vendor.

SECTION 07

The tester: the whole train, from outside

The tester runs once per train, after its last step has landed. It checks the steps together, as the user would, and reports defects as reproductions. It never fixes the product, and its verdict never counts as a review.

- **Against the contract, not the implementation.** Its input is every step's acceptance items and the train's step plan.
- **Its own disposable instance**, built from the commit that carries every step. Destructive tests never touch a shared system.
- **Leave and come back**, every way the platform allows: reload, back button, a new tab, a restart, a lost connection.
- **An edge matrix for every acceptance item**, then **chaos and repeat**: the same path many times, and once at realistic size.
- **Evidence a reviewer can compare**: screenshots, transcripts, logs, in the record.
- **What it cannot run is UNRUN**, with the reason. A blocking finding opens a fix step in the same train, and the tester runs again.

Every screen answers four things

A screen shown to a person is tested for what that person gets from it: **what happened, who did it, why it matters to this reader, and what they can do next**. Everything shown links to its source. A field that does not belong to the kind of item is not rendered.

SECTION 08

The merge gate

- **Merging is a person's act.** The verbs that approve, merge and tear down are denied to every agent at the permission grant, the planner's included. A raw `git merge` is denied too.
- **A recommendation and a decision are two recorded events.** The reviewer produces the first; only a person produces the second.
- **The merged tree must build.** Two changes can merge cleanly and still not compile together. Before a merge is handed to a person, the merged tree is built and statically checked.
- **Commit rules are checked at merge**: subject format, length, English, a body that says why, no AI attribution.
- **Golden files are regenerated on the merged tree**, never taken from one side of a merge.
- **Install is never implicit in a merge.** Whether a new build is installed is the user's choice.

SECTION 09

Measured from the record

Measured on 5 November 2026 on one software project: a large Go and TypeScript code base written by AI agents under Saphan Studio, with a person approving every merge. Every number was counted from the project's repository and its record.

MEASURE	VALUE	METHOD
Commits on the main branch	20,407	since the first commit, four months earlier
Commits in the last 7 / 30 days	1,117 / 7,587	main branch
Merges in the last 7 days	280	merge commits, main branch
Distinct work streams merged in 7 days	194	stream names in merge subjects
Go source files / of them test files	11,453 / 7,649	tracked files on the main branch
Go production lines / test lines	1.34 M / 2.14 M	line count, generated code included
TypeScript files / of them tests	809 / 355	console and clients
Files of recorded test data (goldens, fixtures)	1,244	under testdata/

Review verdicts

VERDICT	COUNT	SHARE
accept	260	33%
accept with conditions	346	44%
reject	183	23%
total	789	100%

Counted from the verdict lines of closed and active work in the record. Older verdicts written in other words (approve, pass, green) are not counted; the three values above are the current vocabulary.

> Two in three changes did not pass review as submitted. That is the point of an independent reviewer: the work that reaches the human approver has already been argued with.

SECTION 10

The same chain, as open-source tools

The testing and review practice is also available as separate open-source tools (Apache 2.0, github.com/saphan-ai). Each one works on its own, without Saphan Studio; inside Saphan Studio they are one chain.

TOOL	WHAT IT ADDS TO TESTING AND REVIEW
Discover	Reads the code before anyone changes it, offline, with a source for every claim, and proves it changed nothing. Its profile tells the agents how the code is built and tested, and proposes the matching skills.
Skills Studio	Checks and signs the skills the implementer and the reviewer are given, so both work to the same verified bar.
Mocks Studio	Stand-in systems (mail, file store, user directory, object store, Kubernetes and more) with a conformance suite, so tests and testers never touch the real system.
Tests Studio	Shows what a test run really proved, in the framework the team already uses: an adapter specification and a conformance suite, so “42 passed” can be checked.
Proven	An evidence-first AI reviewer: it reports a problem only with a failing test or a query and its rows, in a signed evidence pack.

SECTION 11

Code quality against published baselines

For a measurement of code quality against published industry baselines (test coverage, documented API, comment density, complexity), see the public report on Proven, an open-source AI code reviewer built by agents under Saphan Studio:

Six days with agents, 14 to 20 months for people — saphan.ai/research/proven-effort

Every number in that report can be checked in its public repository, github.com/saphan-ai/proven.

SECTION 12

What this document does not claim

- The numbers describe one project. They are not a defect rate in production and not a promise for another code base.
- Line counts include generated code. The ratio of test lines to production lines is a measure of effort, not of coverage.
- A review recommends; it can be wrong. The human approval, the tester and the record that keeps every step are there because no single check is enough.

SECTION 13

The Saphan ecosystem and this document set

Saphan Studio is one product in a small ecosystem. The tools around it are open source under Apache 2.0 and each works on its own; together they cover a change from the first read of the code to the evidence that it is right.

`Discover reads the code` → `Skills Studio checks the skills` →
`Saphan Studio runs the agents under a person's approval` → `Mocks Studio stands in for real systems` →
`Tests Studio and Proven judge the result`

TOOL	ITS PART
Saphan Studio saphan.ai	The control plane: signed orders, confined agents, independent review, human approval, one signed record.
Discover github.com/saphan-ai/discover	Reads a codebase before anyone changes it, with a source for every claim; proposes the skills that fit it.
Skills Studio github.com/saphan-ai/skills-studio	Checks and signs the skills agents are given, so anyone can verify them again.
Mocks Studio github.com/saphan-ai/mocks-studio	Stand-in systems agents and tests work against instead of the real ones.
Tests Studio github.com/saphan-ai/tests-studio	Shows what a test run really proved, in the framework the team already uses.
Proven github.com/saphan-ai/proven	An AI code reviewer that reports a problem only with evidence.

The document set

DOCUMENT	WHAT IT ANSWERS
Saphan Studio for developers and DevOps	start here: what Saphan Studio is and how the pieces fit
Skills and craft	what every agent carries into the work
Testing and review (this document)	how a change earns its merge
Orchestration rules	a deterministic engine, model work inside it
Security	what an agent can reach, and what you can prove afterwards

For partners: a founding customer onboarding guide and an Enterprise continuity plan, at saphan.ai/for-partners.

All documents: saphan.ai/product · Code quality against published baselines: saphan.ai/research/proven-effort · Operator documentation: docs.saphan.ai · Contact: hello@saphan.ai